Informe del Trabajo Integrador

Programación con Objetos II

1er cuatrimestre, 2024

Integrantes

- Lara Noelí Cardozo: cardozo.lara.edu@gmail.com
- Malena Sciutto: male.sciutto02@gmail.com
- Thiago Ezequiel Sofarelli: sofarelli.thiago10@gmail.com

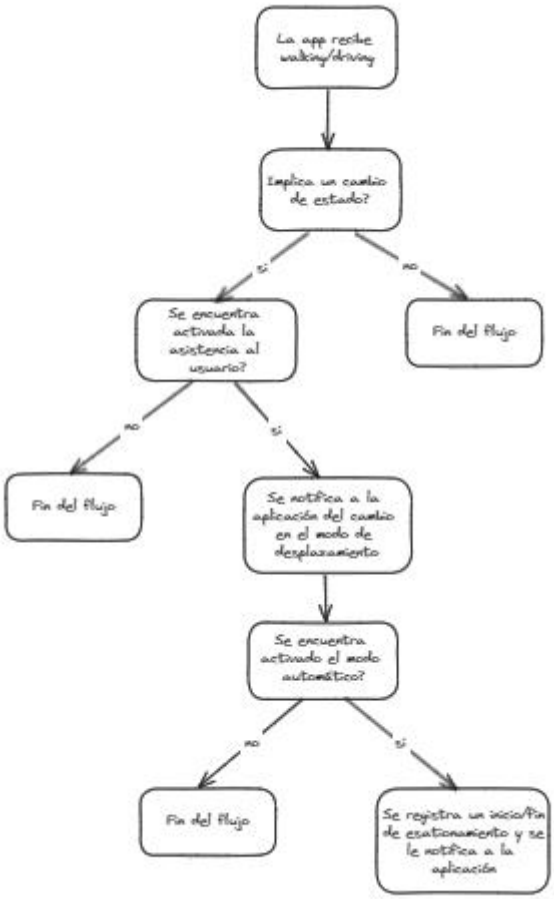
Decisiones de diseño

Al desarrollar nuestro trabajo y con el objetivo de crear código reutilizable y capaz de ser extendido para cumplir futuras necesidades, seguimos los principios SOLID, que expresan entre otras cosas la importancia de la clara delimitación de responsabilidades de cada clase, los beneficios de desarrollar código abierto a la extensión pero cerrado a la modificación y de tener una clara jerarquía de clases.

La primera de las decisiones de diseño que tomamos para esto fue la separación de responsabilidades del Sistema de Estacionamiento medido. Como notamos que las tareas de gestionar inicios, fines y vigencias de estacionamientos constituían gran parte de las responsabilidades del SEM y eran claramente agrupables, creamos una nueva clase “Gestor de Estacionamientos” para ocuparse de ellas.

Por otro lado, intentamos aprovechar similitudes y utilizar polimorfismo para el manejo de gran parte de las clases de nuestro trabajo. Esto se puede ver en la subclasificación de clases como RegistroDeCompra o Estacionamiento.

Por último, delegamos el comportamiento de la App de estacionamiento en sus colaboradores de estado, asistencia y modo, utilizando los patrones de diseño que presentaremos en la siguiente sección para crear un flujo fácil de modificar y extender, y que queda expresado a continuación:



Patrones de diseño

State

Utilizamos el patrón State para gestionar el estado del usuario de nuestra app, que puede tomar dos valores: Estacionado o Manejando. Esta decisión de diseño hizo posible la detección de un cambio de estado para los usuarios con la app en modo automático y con la asistencia al usuario activada, además de prevenir, por ejemplo, que un usuario con un estacionamiento iniciado intente iniciar otro antes de terminar el primero.

En cuanto a los roles de este patrón, nuestra app cumplió el rol de Context, el Estado de State, y las clases Estacionado y Manejando, de Concrete States.

Strategy

El patrón de Strategy fue útil para manejar el modo de la aplicación (manual o automático), así como la asistencia elegida por el usuario (activada o desactivada). Permite delegar comportamiento a los colaboradores de la App, pero al contrario del State, es ésta misma la que puede cambiar entre uno y otro modo.

La aplicación una vez más cumplió el rol de Context, mientras que las interfaces ModoApp y AsistenciaAlUsuario fueron los Strategies. Las clases Automatico/Manual y Activada/Desactivada son las Concrete Strategies de ModoApp y AsistenciaAlUsuario, respectivamente.

Observer

Utilizamos el patrón Observer para notificar a las entidades suscriptas al SEM de cambios como incio y fin de estacionamientos y recarga de crédito.
El SEM en este caso cumplió el rol de sujeto concreto, con la interfaz ObserverEstacionamiento haciendo de sujeto abstracto. La interfaz Suscriptor, por su lado es el observer abstracto, y cualquier entidad que la implemente, un observer concreto.